Lecture 12: Linearly constrained nonlinear optimization

Michael Patriksson

26 February 2004

Feasible-direction methods

- Consider the problem to find

$$f^* = \text{minimum } f(\boldsymbol{x}), \tag{1a}$$
$$\text{subject to } \boldsymbol{x} \in X, \tag{1b}$$

$X \subseteq \mathbb{R}^n$ non-empty, closed and convex set; $f : \mathbb{R}^n \mapsto \mathbb{R}$ is C^1 on X.

- Most methods for (1) manipulate the constraints defining X; in some cases even such that the sequence $\{\boldsymbol{x}_k\}$ is infeasible until convergence. Why?
- Consider a constraint "$g_i(\boldsymbol{x}) \leq b_i$," where g_i is nonlinear.

- Checking whether $\boldsymbol{p}$ is a feasible direction at $\boldsymbol{x}$, or what the maximum feasible step from $\boldsymbol{x}$ in the direction of $\boldsymbol{p}$ is, is very difficult.
- For which step length $\alpha > 0$ does it happen that $g_i(\boldsymbol{x} + \alpha\boldsymbol{p}) = b_i$? This is a nonlinear equation in α!
- Assuming that X is polyhedral, these problems are not present.

Feasible-direction descent methods

Step 0. Determine a *starting point* $\boldsymbol{x}_0 \in \mathbb{R}^n$ such that $\boldsymbol{x}_0 \in X$. Set $k := 0$.

Step 1. Determine a *search direction* $\boldsymbol{p}_k \in \mathbb{R}^n$ such that $\boldsymbol{p}_k$ is a feasible direction.

Step 2. Determine a *step length* $\alpha_k > 0$ such that $f(\boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k) < f(\boldsymbol{x}_k)$ and $\boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k \in X$.

Step 3. Let $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$.

Step 4. If a *termination criterion* is fulfilled, then stop! Otherwise, let $k := k + 1$ and go to Step 1.

Notes

- Similar form as the general method for unconstrained optimization.
- Just as *local* as methods for unconstrained optimization.
- Search directions typically based on the approximation of f—relaxation!
- Line searches similar; note the maximum step.
- Termination criteria and descent based on first-order optimality (remember the unconstrained condition that $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}^*$ holds).

LP-based algorithm, I: The Frank–Wolfe method

- The Frank–Wolfe (1952) method is based on a first-order approximation of f around the iterate $\boldsymbol{x}_k$. This means that the relaxed problems are LPs, which can then be solved by using the Simplex method.
- Remember the following first-order condition [Proposition 4.20(b)]: *If $\boldsymbol{x}^* \in X$ is a local minimum of f on X then*

$$\nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*) \geq 0, \qquad \boldsymbol{x} \in X, \tag{2}$$

holds.

- Remember also the following equivalent statement:

$$\underset{\boldsymbol{x} \in X}{\text{minimum}}\ \nabla f(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}^*) = 0. \tag{3}$$

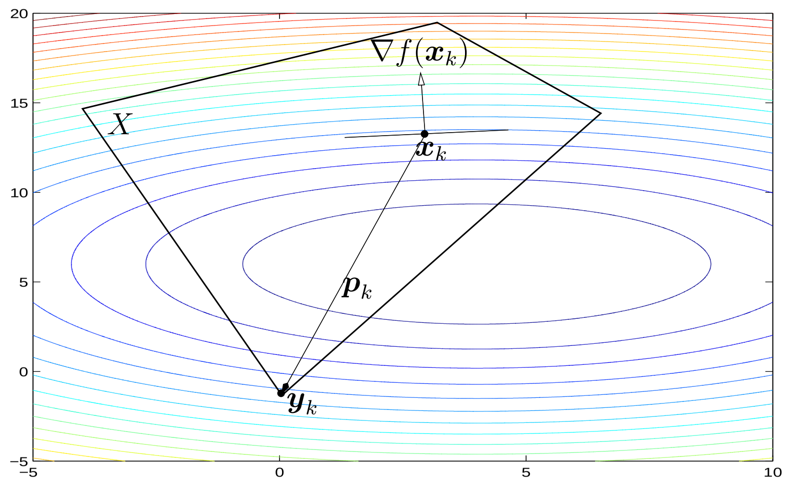
- Follows that if, given an iterate $\boldsymbol{x}_k \in X$,

$$\underset{\boldsymbol{y} \in X}{\text{minimum}}\ \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}_k) < 0,$$

  and $\boldsymbol{y}_k$ is a solution to this LP problem, then the direction of $\boldsymbol{p}_k := \boldsymbol{y}_k - \boldsymbol{x}_k$ is a feasible descent direction with respect to f at $\boldsymbol{x}$.
- The search direction is towards an extreme point [one that is optimal in the LP over X with costs $\nabla f(\boldsymbol{x}_k)$].
- This is the basis of the Frank–Wolfe algorithm.

- Note: We must assume that X is bounded in order to ensure that the LP always has a finite solution. The algorithm can in fact be extended to allow for unbounded solutions to the LP, and thereby extending the Frank–Wolfe method for general polyhedra; the search directions then are either towards an extreme point (finite solution to LP) or in the direction of an extreme ray of X (unbounded solution to LP).

The search-direction problem



Algorithm description, Frank–Wolfe

Step 0. Find $\boldsymbol{x}_0 \in X$, for example any extreme point in X. Set $k := 0$.

Step 1. Find a solution $\boldsymbol{y}_k$ to the problem to

$$\underset{\boldsymbol{y} \in X}{\text{minimize}} \; z_k(\boldsymbol{y}) := \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}_k). \qquad (4)$$

Let $\boldsymbol{p}_k := \boldsymbol{y}_k - \boldsymbol{x}_k$ be the search direction.

Step 2. Approximately solve the problem to minimize $f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k)$ over $\alpha \in [0, 1]$. Let α_k be the step length.

Step 3. Let $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$.

Step 4. If, for example, $z_k(\boldsymbol{y}_k)$ or α_k is close to zero, then terminate! Otherwise, let $k := k + 1$ and go to Step 1.

Convergence

- Theorem 12.1: *Suppose that $X \subseteq \mathbb{R}^n$ is a non-empty, bounded polyhedron, and that the function f is in C^1 on X. Suppose that in Step 2 of the Frank–Wolfe algorithm, we either use an exact line search or the Armijo step length rule. Then, the sequence $\{\boldsymbol{x}_k\}$ is bounded, $\{f(\boldsymbol{x}_k)\}$ is descending, and every limit point (at least one exists) is stationary; further, the sequence $\{z_k(\boldsymbol{y}_k)\} \to 0$.*
  If f is convex on X, then every limit point is globally optimal. ■

The convex case: Lower bounds

- Remember the following characterization of convex functions in C^1 on X [Theorem 3.44(a)]:
  f is convex on X $\Longleftrightarrow$
  $$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}), \quad \textit{for all } \boldsymbol{x}, \boldsymbol{y} \in X.$$
- Suppose f is convex on X. Then, $f(\boldsymbol{x}_k) + z_k(\boldsymbol{x}_k) \leq f^*$ (lower bound, LBD), and $f(\boldsymbol{x}_k) + z_k(\boldsymbol{x}_k) = f^*$ if and only if $\boldsymbol{x}_k$ is globally optimal.
- Utilize the lower bound as follows: we know that $f^* \in [f(\boldsymbol{x}_k) + z_k(\boldsymbol{x}_k), f(\boldsymbol{x}_k)]$. Store the best LBD, and check in Step 4 whether $[f(\boldsymbol{x}_k) - \text{LBD}]/|\text{LBD}|$ is small, and if so terminate.

Final comments

- Frank–Wolfe uses linear approximations—works best for almost linear problems.
- For highly nonlinear problems, the approximation is bad—the optimal solution may be far from an extreme point.
- In order to find a near-optimum requires many iterations—the algorithm is slow.
- Another reason is that the information generated (the extreme points) are forgotten. Even if we keep the linear subproblems, we can do better by storing and utilizing this information.

LP-based algorithm, II: Simplicial decomposition

- Remember the Representation Theorem 9.9 (special case for bounded polyhedra): *Let* $P = \{\, \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b};\ \boldsymbol{x} \geq \boldsymbol{0}^n \}$, *be non-empty and bounded, and* $V = \{\boldsymbol{v}^1, \dots, \boldsymbol{v}^K\}$ *be the set of extreme points of* P. *Every* $\boldsymbol{x} \in P$ *can be represented as a convex combination of the points in* V, *that is,*

$$\boldsymbol{x} = \sum_{i=1}^{K} \alpha_i \boldsymbol{v}^i,$$

for some $\alpha_1, \dots, \alpha_k \geq 0$ *such that* $\sum_{i=1}^{K} \alpha_i = 1$. ■

- The idea behind the Simplicial decomposition method is to generate the extreme points $\boldsymbol{v}^i$ which can be used to describe an optimal solution $\boldsymbol{x}^*$, that is, the vectors $\boldsymbol{v}^i$ with positive weights α_i in

$$\boldsymbol{x}^* = \sum_{i=1}^{K} \alpha_i \boldsymbol{v}^i.$$

- The process is still iterative: we generate a "working set" $\mathcal{P}_k$ of indices i, optimize the function f over the convex hull of the known points, and check for stationarity and/or generate a new extreme point.

Algorithm description, Simplicial decomposition

Step 0. Find $\boldsymbol{x}_0 \in X$, for example any extreme point in X. Set $k := 0$. Let $\mathcal{P}_0 := \emptyset$.

Step 1. Let $\boldsymbol{y}_k$ be a solution to the LP problem (4). Let $\mathcal{P}_{k+1} := \mathcal{P}_k \cup \{k\}$.

Step 2. Let $(\mu_k, \boldsymbol{\nu}_{k+1})$ be an approximate solution to the *restricted master problem* to

$$\begin{aligned} &\underset{(\mu,\boldsymbol{\nu})}{\text{minimize}} \quad f\left(\mu \boldsymbol{x}_k + \sum_{i\in\mathcal{P}_{k+1}} \nu_i \boldsymbol{y}_i\right), && \text{(5a)} \\ &\text{subject to} \quad \mu + \sum_{i\in\mathcal{P}_{k+1}} \nu_i = 1, && \text{(5b)} \\ &\qquad\qquad \mu, \nu_i \geq 0, \qquad i \in \mathcal{P}_{k+1}. && \text{(5c)} \end{aligned}$$

Step 3. Let $\boldsymbol{x}_{k+1} := \mu_{k+1}\boldsymbol{x}_k + \sum_{i\in\mathcal{P}_{k+1}} (\boldsymbol{\nu}_{k+1})_i \boldsymbol{y}^i$.

Step 4. If, for example, $z_k(\boldsymbol{y}_k)$ is close to zero, or if $\mathcal{P}_{k+1} = \mathcal{P}_k$, then terminate! Otherwise, let $k := k+1$ and go to Step 1.

- This basic algorithm keeps all information generated, and adds one new extreme point in every iteration.
- An alternative is to drop columns (vectors $\boldsymbol{y}_i$) that have received a zero weight, or to keep only a maximum number of vectors. (Stated in the Notes.)
- Special case: maximum number of vectors kept $= 1 \Longrightarrow$ the Frank–Wolfe algorithm!
- We obviously improve the Frank–Wolfe algorithm by utilizing more information.

Convergence

- Based on the fact that it does at least as well as the Frank–Wolfe algorithm.
- Convergence is finite if the restricted master problems (RMPs) are solved exactly, and the maximum number of vectors kept is at least as many as are needed to span $\boldsymbol{x}^*$.
- Much more efficient than the Frank–Wolfe algorithm in practice.
- We can solve the RMPs efficiently, since they are almost unconstrained.

An illustration of FW vs. SD

- A large-scale non-linear network flow problem which is used to estimate traffic flows in cities.
- The model is over the small city of Sioux Falls in North Dakota, whose representation has 24 nodes, 76 links, and 528 pairs of origin and destination.
- Three algorithms for the RMPs were tested—a Newton method and two gradient projection methods (see the next section). A MATLAB implementation.
- Remarkable difference—The Frank–Wolfe method suffers from very small step length being taken.

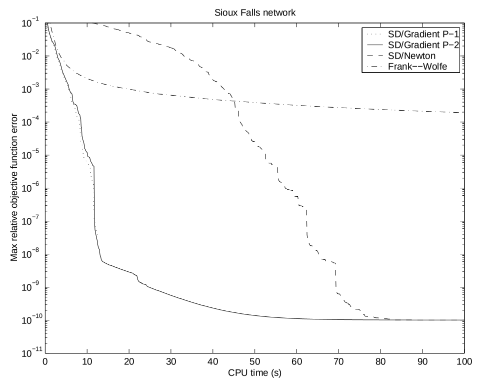
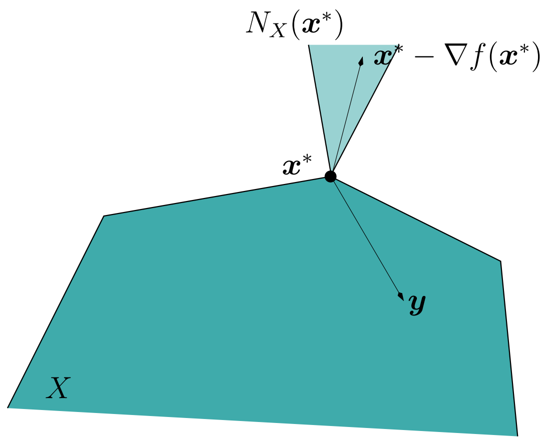


Figure 1: The performance of DSD vs. FW on the Sioux Falls network.

QP-based algorithm: The gradient projection algorithm

- The gradient projection algorithm is based on the projection characterization of a stationary point (Section 4.4): $\boldsymbol{x}^*$ is a stationary point if and only if

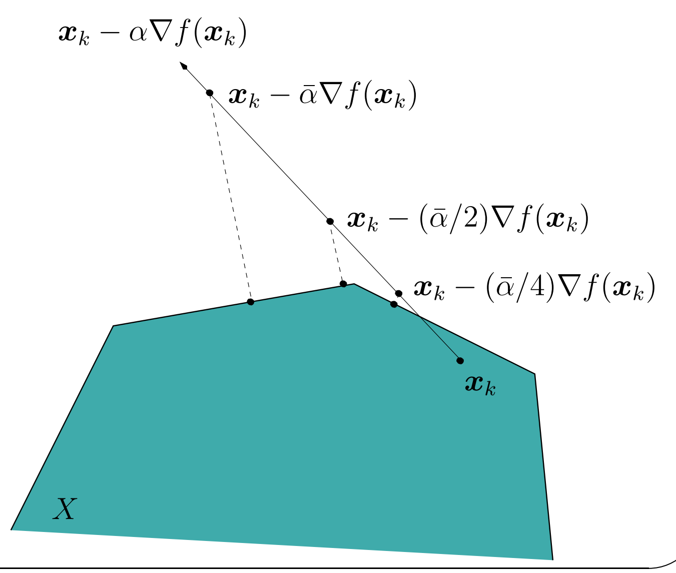
$$\boldsymbol{x}^* = \mathrm{Proj}_X[\boldsymbol{x}^* - \nabla f(\boldsymbol{x}^*)].$$



- Let $\boldsymbol{p} := \mathrm{Proj}_X[\boldsymbol{x} - \alpha\nabla f(\boldsymbol{x})] - \boldsymbol{x}$, for any $\alpha > 0$. Then, if and only if $\boldsymbol{x}$ is non-stationary, $\boldsymbol{p}$ is a feasible descent direction of f at $\boldsymbol{x}$.
- The gradient projection algorithm is normally stated such that the line search is done over the *projection arc*, that is, we find a step length α_k for which

$$\boldsymbol{x}_{k+1} := \mathrm{Proj}_X[\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k)], \qquad k = 1, \ldots . \qquad (6)$$

has a good objective value. Use the Armijo rule to determine α_k:


$\boldsymbol{x}_k - \alpha\nabla f(\boldsymbol{x}_k)$
$\boldsymbol{x}_k - \bar{\alpha}\nabla f(\boldsymbol{x}_k)$
$\boldsymbol{x}_k - (\bar{\alpha}/2)\nabla f(\boldsymbol{x}_k)$
$\boldsymbol{x}_k - (\bar{\alpha}/4)\nabla f(\boldsymbol{x}_k)$
$\boldsymbol{x}_k$
X

Convergence

- Theorem 12.3 (simplified): *Suppose that $X \subseteq \mathbb{R}^n$ is non-empty, compact and convex. Consider the iterative algorithm defined by the iteration* (6)*, where the step length α_k is determined by the Armijo step length rule along the projection arc. Then, the sequence $\{\boldsymbol{x}_k\}$ is bounded, the sequence $\{f(\boldsymbol{x}_k)\}$ is descending, lower bounded and therefore has a limit, and every limit point of $\{\boldsymbol{x}_k\}$ is stationary.* ■
- Gradient projection becomes steepest descent with Armijo line search when $X = \mathbb{R}^n$!
- Convergence arguments similar to steepest descent one.

Quadratic subproblems—how are they solved?

- State the KKT conditions for the strictly convex QP problem which determines the projection.
- Add slack variables.
- Result: A system of linear inequalities and equalities plus two sets of complementarity conditions of the form $x_j\nu_j = 0$ and $s_i\mu_i = 0$.
- Set up a Phase I problem for the linear inequality system, and treat the complementarity conditions implicitly, as follows: if x_j (respectively, v_j) is a basic variable, then v_j (respectively, x_j) must not be an entering variable. Same for the pair (s_i, μ_i).